EFFICIENT SUPPORT OF STATISTICAL OPERATIONS

by

Setrag N. Khoshafian
Douglas M. Bates
David J. DeWitt

Computer Sciences Technical Report #582

February 1985

Efficient Support of Statistical Operations

by

Setrag N. Khoshafian
MCC, Austin, Texas

Douglas M. Bates
Statistics Department, University of Wisconsin-Madison

David J. DeWitt
Computer Sceiences Department, University of Wisconsin-Madison

This research was partially supported by the Department of Energy under contract #DE-AC02-81ER10920.

ABSTRACT

Most research in statistical databases has concentrated on retrieval, sampling, and aggregation type statistical queries. Data management issues associated with computational statistical operations have been ignored. As a first step towards integrating database management support of statistical operations, we have analyzed the performance of $X'X$, the QR decomposition, and the Singular Value Factorization. Alternative implementation strategies with respect to the relational and transposed storage organizations are developed. Implementation strategies corresponding to vector building block, vector-matrix, and direct algorithms with explicit buffer management are compared in terms of efficiency in performance.

Efficient Support of Statistical Operation

by

Setrag N. Khoshafian, Douglas M. Bates, and David J. DeWitt

1. Introduction

Recently, the data management issues associated with accessing and processing large statistical databases has become an active research area. A number of papers (e.g. [TURN79, SHOS82, DENN83, ROWE83] etc.) and projects (e.g. the ALDS project [THOM83], SEEDIS [McCA82], SAM* [STAN83] etc.) have attempted to identify some of the data access, retrieval, and management characteristics of large statistical databases. Some of these research efforts have proposed, and sometimes implemented, innovative solutions to non-conventional database management issues.

The characterization and the proposed solutions for statistical database management systems can be classified into two interdependent categories:

(1) those pertaining to the statistical data sets themselves

(2) those pertaining to the manipulation of the statistical data sets, or, statistical analysis.

For the first category, it has been observed that statistical data sets tend to be large and complex. Statistical data sets inevitably contain several types of missing observations. Structured missing data values (or sparse data) are also common in large statistical databases. Another characteristic of statistical data sets is the distinction between category and summary attributes. Finally, it has been observed that statistical data sets tend to be more or less static. Observations over large samples do not tend to change as often as transactions for corporate databases. This is also a characteristic of statistical analysis.

It has been observed that the process of statistical analysis passes through two phases. The **exploratory** phase and the **confirmatory** phase [TUKE77, BORA82]. In the exploratory phase

the analyst attempts to obtain a "feel" for the data set by editing and browsing through the database, extracting samples and subsets and performing some initial hypothesis testing. From a data management point of view, the problem is the generation and the dynamic maintenance of these samples and subsets. Later, after this initial exploratory phase, the analyst tries to "confirm" the hypotheses over larger data sets. In this confirmatory phase, the entire data set is manipulated. However, it has been observed that most of the observations and only a few of the attributes are retrieved in this phase.

The direct and efficient support of complex statistical operations (such as linear least squares problems) is the next step towards efficient integration of statistical capabilities in a DBMS. Although statistical queries could be handled by statistical packages, we shall demonstrate the resulting drawbacks of such an approach, especially when very large data sets are involved. Instead we advocate adoption of an integrated approach where the SDBMS supports both retrieval and computational queries.

As a first step towards developing an integrated system, we identify and analyze the performance of three important statistical operations: $X'X$, the QR decomposition, and the Singular Value Factorization. An important underlying assumption is that the data sets are very large. For our purposes we assume data sets are n by p, where $n >> p$. Moreover, we assume results which are of the order $O(p^2)$ or less could be maintained in main storage. Therefore, secondary storage overheads are caused only by data sets (either temporary or base) which are of the order O(n) or more.

For the efficient support of the three operations $X'X$, QR, and SVF, we have considered and compared the performances with respect to the two most common secondary storage organizations: transposed and relational, as well as several alternative implementation strategies. The alternative implementations correspond to vector building block, vector-matrix, and direct algorithms for the computational operations.

The paper is organized as follows: Section 2 motivates and explains the implications of supporting statistical operations in the DBMS. Section 3 demonstrates the importance of large linear

least squares problems, and justifies the choice of three operations X'X, QR, and SVF as a first step towards an integrated system. Section 4 describes the performance model. In Sections 5, 6, and 7 we specify algorithms for the three operations, as well as analytical evaluations of different implementation strategies. Section 8 presents our conclusions and suggestions for future research directions.

2. DBMS Support for Statistical Operations

The data management issues associated with complex, computational, or analytical statistical queries (such as multiple linear regression, analysis of variance, canonical correlation etc.), have not been previously considered by research efforts in statistical databases. Previously, the underlying assumption has been that computational or analytical statistical queries are handled by statistical packages rather than by the statistical database management systems. Therefore, for these type of queries the statistical database management system provides and maintains interfaces to one or more statistical packages. A case in point is the the ALDS project [THOM83]. The data sets and the meta-data are organized in self describing transposed files and the system provides an interface to the statistical package Minitab [RYAN76].

There are four main problems with this approach. First, as pointed out in [BORA82], the data sets manipulated by some statistical packages cannot exceed the virtual memory size of the machine. Second, the buffer management strategy employed is generally that of the operating system, which often uses a global buffer management strategy and does not take into consideration the particular data access patterns of the statistical operations being executed. Third, the data sets that are manipulated by the statistical package must be copied into the workspace of the statistical package, processed and, sometimes, the resulting data sets must be copied back into the database. Finally, usually algorithms used to implement the operations do not attempt to minimize the number of secondary storage accesses.

The main form of improvement in the I/O performance of the interface approach has been in terms of enhanced (paginated) algorithms which attempt to reduce the number of page faults. Several programming techniques for processing matrix operations in paginated (virtual memory) environments have been proposed (e.g. [MOLE72], [ELSH74], [McKE69], [TRIV77] etc.). These algorithms attempt to optimize the I/O performance of the matrix operations, for a given page replacement algorithm (in most cases LRU). The goal is to improve page locality, and hence reduce the number of page faults. However, besides operating in global buffer management and virtual memory environments, these algorithms ignore the performance implications associated with the concurrent operation of I/O and processing subsystems. Below we shall demonstrate how our integrated approach differs fundamentally from these enhanced interface algorithms, and propose a methodology for supporting the computational methods at the internal ANSI/SPARC level of a DBMS architecture.

The alternative to the interface approach is to have an integrated system and let the statistical database management system support both retrieval and computational queries. The primary difficulty with the integrated approach is the large (and growing) number of techniques that a statistician can choose from when analyzing a data matrix X (e.g. principal component analysis, discriminant analysis, etc.). Moreover, better and alternative analytical techniques are continually being developed. Faced with this problem, we adopted a simplistic approach. We decided that the first step towards an integrated system was the detailed analysis of alternative computational methods for one important analytical technique. We chose to concentrate on the method of fitting linear statistical models. This is also called multiple linear regression analysis. In Section 3 we provide additional justification for this choice.

DBMS support for the computational methods can be modeled in terms of three categories:

(1) Implementations at three abstraction levels: corresponding to vector building block, vector matrix, and direct algorithms. These are further explained in Section 4.1.

(2) Secondary storage organizations: the two most common type of storage organizations in existing DBMS's are the fully transposed and the relational organizations. A qualitative description and the tradeoffs of these two strategies is presented in Section 4.2.

(3) Alternative buffer management algorithms which explictily specify: (a) the number of subdivisions of the main storage; (b) the page replacement strategy; (c) the concurrent execution of the I/O and processing subsystems.

The partitioning of the main storage depends upon the number of active columns (explained in Section 4), and the main storage size. For the page replacement policy our algorithms incorporate page Read, Retain, Free, and Write primitives. The concurrent execution of the I/O and processing subsystems is indicated by preceding each concurrent process with a "$." Typically each algorithm involves either two or three concurrent processes. The two-process algorithms correspond to operations which involve only reads (i.e. small outputs, such as X'X). The three-process algorithms correspond to operations which read, update, and write large submatrices (such as the QR decomposition).

It is to be emphasized that after fixing a particular storage organization, and an abstraction level, we still have several alternative strategies for actual implementation, and we have expressed these alternatives in terms of explicit buffer management algorithms. We have developed closed form cost equations in terms of the system and data parameters. This enabled us to analyze the performance of the algorithms while varying the main storage size, the number of active columns, and the processing speed. Consequently, we are able to draw some important conclusions regarding the implications of DBMS support for statistical operations.

3. Large Linear Least Squares Problems

The statistician's goal in data analysis can be simply to produce arithmetic (means, standard deviations, etc.) or graphical (histograms, etc.) summaries of attributes and these aggregate operations are often supported in a DBMS. However, a more realistic and useful analysis of data will typically involve determining and modeling relationships between attributes in a data set. There are many different modeling techniques including a variety of regression techniques (linear, stepwise, robust, ridge, iterative reweighted, non-linear), analysis of variance and covariance, canonical correlation, principal component analysis, discriminant analysis, categorical data analysis, clustering techniques, projection-pursuit methods, time-series analysis, and so on.

The primary difficulty with the proposed integrated approach to supporting these techniques in a DBMS is that there are so many and new ones are continually being created. Therefore, it is

difficult to define a "closed" or functionally complete statistical database management system.

Fortunately we can create another level of abstraction under the level of the particular statistical techniques. Many of the techniques rely on one or more applications of a basic statistical calculation, the most common of which is the solution of a linear least squares problem.

The importance of this basic statistical calculation was illustrated at the PARVEC workshop on very large least squares problems [RICE84]. This workshop brought together computer scientists who work with supercomputers and scientists who use supercomputers in applications areas so they could discuss the state of the art in very large least squares problems and determine areas where research is most needed. These scientists described applications in areas such as geodetic surveys, molecular structures, gravity, and partial differential equations. Increasingly these applications areas depend on being able to fit very large nonlinear models to data sets using least squares. A basic building block for these calculations is the linear least squares calculation.

The linear least squares calculation is a basic step in many statistical calculations. The calculations for linear regression analysis and the analysis of variance, two of the most common statistical modeling techniques, are a single linear least squares problem. The Generalized Linear Models of Nelder and Wedderburn [McCU83], logistic regression models, and nonlinear regression models all use the linear models calculations as a basic iterative step.

In addition, most multivariate analysis methods such as canonical correlations, principal component analysis, and discriminant analysis use the same matrix calculations as the linear least squares problems so implementing this basic statistical calculation efficiently is helpful for a wide variety of statistical techniques.

There are several alternative computational methods for linear least squares problems. Either the symmetric matrix $X^t X$ is formed [SEBE77, KENN80] and the solution proceeds with this matrix, or the matrix X is decomposed and the solution is calculated from the factors of X. The most common decompositions of a data matrix X are the upper triangularization of X (through Givens rotations or Householder transformations [GOLU73, SEBE77, KENN80,

DONG79]) and the Singular Value Decomposition [STEW73, DONG79, WILL71, KENN80, CUPP81, MAND82]. Therefore, a list of matrix operations which covers the evaluation of X' X and the most common decompositions of X will provide the computational tools of the linear least squares problem. The statistical technique can be viewed as an application program.

There are fundamental differences in the X' X and decomposition approaches and we shall see this reflected in the performance of algorithms. Using X' X is conceptually simpler and results in a row-oriented algorithm. That is, the data can be processed a row (or case) at a time and each case is only required once. This makes the detection of missing data and modifications for the missing data straightforward. In contrast, the decomposition methods are more complicated and are column oriented so modifications for missing data are more difficult. Also, parts of the data matrix must be accessed many times since the columns are repeatedly modified. The advantage of the decomposition methods are that they are more stable numerically and they can provide secondary information which is not directly available from X' X.

Numerical analysts have examined the numerical properties and numerical efficiency of these methods extensively. See, for example, the discussion in [STEW73] or [DONG79]. However, these methods have not been examined from the viewpoint of data flow or I/O. An interesting conclusion of the PARVEC workshop [RICE84, p.13] is that non-numerical bottlenecks such as I/O and page-thrashing can be as important to the overall speed of the computation as the organization of the arithmetic and the use of vectorization. This conclusion is reinforced here for we also find that the low-level I/O considerations are comparable in importance to the numerical considerations.

4. Performance Model

4.1. Implementations at Three Abstraction Levels

While this is the first time that the secondary storage problems of the computational methods X' X, the QR decomposition, and the Singular Value Factorization have been con-

sidered, other research studies have attempted to identify the data management issues of different classes of computational operations. For example, [PERR81] considers the problem of solving triangular or banded systems of linear equations. The author proposes a number of secondary storage methods and parallel algorithms and compares his algorithms with the execution of the same operation in a virtual memory environment (with a LRU page replacement strategy). In [BELL83], the author characterizes several storage structures that are used in solving weather forecasting and other types of prediction methods.

In this paper we analyze the performance of three important computational methods commonly used in multiple linear regression, with respect to three abstraction levels. In the following sections we identify the three abstraction levels for implementing these computational methods and briefly describe the relative merits of each.

4.1.1. Vector Building Block

The first abstraction level attempts to implement the computational methods through vector operations only. This approach has been used by numerical analysts [DONG79, LAWS79] in isolating a set of vector operations called the BLAS (Basic Linear Algebra Operations) which are coded in a assembly language to speed execution. As an example of the vector operations approach, the computational method $X^t X$ can be implemented as a set of inner products. The advantage of this approach is that a relatively small set of vector operations will enable us to implement all the computational methods. Our results demonstrate, the disadvantage of this approach is the I/O performance, since the set of vector operations which implements a computational method is completely unaware of the computational method's overall reference pattern.

4.1.2. Vector-Matrix

The approach attained by the second abstraction level is intermediate between the approaches for levels one and three. With the vector-matrix approach, the computational methods are implemented through vector-matrix operations. The multiplication of a vector by a matrix and the Householder transformation are two examples of vector-matrix operations. As we

shall see, in most applications a vector-matrix implementation of an operation will involve considerably fewer page transfers than a vector building block implementation of the same operation. The same objection, however, which was raised for the vector building block approach still holds. It is still feasible that a more direct implementation of a computational method provides better performance than an implementation through vector-matrix building blocks.

4.1.3. Direct Algorithms

The final strategy is direct implementation of each of the computational methods considered. The goal is to develop optimal algorithms: optimal in secondary storage reference and also in overall total execution time, which can then be used as a basis for comparison with the first two strategies. However, there must be a direct implementation for each computational method. If a new operation is introduced, the database management system software must be extended to include it.

4.2. Storage Organization

One of the primary goals of this research was to compare the performance of the computational methods with respect to two secondary storage organizations: the relational and the transposed. Most statisticians prefer to view their data as flat files. Therefore the conventional relational secondary storage organization where the n-tuples of each relation are stored record-wise appears to be the natural choice. However, there are at least three reasons why such a secondary storage organization may not be suitable for storing a large statistical data set:

(1) Data compression, through encoding or run length compression is not easily supported [TURN79, EGGR81, KHOS84].

(2) Statistical operations usually encompass most of the observations and only a few of the attributes [TURN79, SHOS82, KLEN83]. With the relational organization all the attributes are retrieved. For large databases this can have serious performance implications.

(3) Finally, it is not uncommon in statistical databases to create or delete attributes. Relational (or corporate) databases on the other hand are tailored to tuple operations (insertion, deletion, and modification of tuples). Many relational database systems do not allow dynamic creation and deletion of attributes.

These drawbacks suggest an alternative storage organization, namely the **transposed** file organization. Transposed file organizations are commonly used in statistical

databases. For example RAPID [TURN79], the ALDS project [BURN81], IMPRESS [MEYE69], and PICKLE [BAKE76] all use the transposed file organization.

There have been a number of studies analyzing transposed files for relational database queries ([HOFF76, BATO79]). However, there have been no analytical studies comparing the performance of the relational and transposed secondary storage organizations for operations on statistical databases. In this paper we compare the execution times of X'X, the QR decomposition, and the Singular Value Factorization (SVF) with respect to both transposed and relational storage organizations. One of the parameters that we varied is the number of **active** columns. The active columns are those columns considered by the analyst. For example, although the original data matrix of observations might contain 100 attributes, the analyst might be interested in fitting a least squares model with only five of the attributes. The remaining 95 attributes are ignored and the 5 attributes constitute the active columns.

For the transposed storage organization, we considered vector building block, vector-matrix building block, and direct algorithms. For the relational storage organization only direct algorithms were considered. The description and concurrent, PASCAL-like algorithms are given only for the direct algorithms with the transposed organization. See [KHOS84] for detailed descriptions of the other configurations.

4.3. System and Model Parameters

The system analyzed consists basically of an I/O subsystem and a processing subsystem. The I/O subsystem is composed of one or more magnetic disk drives that have direct access to the main storage of the processing subsystem. The processing subsystem is composed of a main storage module[1] and a processor dedicated to numeric calculations. A key point is the simplicity of the underlying architecture. Our objective is to obtain parametrized cost equations for a general architecture. This would enable the designers of the database management system to easily modify the parameter values for their particular architectures.

---

[1] which has bandwidth to support concurrent access by both the processor and I/O subsystems

4.3.1. Disk Parameters

The basic mass storage device is assumed to be a disk. The values of the disk parameters are those of the IBM 3350 disk drive [IBM 77]. A page (the unit of transfer between the disk and the main memory) is assumed to be the size as a disk track.

BSIZE	page size	19069 bytes
DCYL	number of pages per cylinder	30 pages
Tio	page read/write time	25.0 ms.
Tdac	average access time	25.0 ms.
Tsk	time to seek 1 track	10.0 ms.
Tmv	time to move to next track after an initial move	0.07 ms.

We have assumed that for those operations that only retrieve data, only one disk drive is used. On the other hand, for those operations which produce temporary files we shall assume the existence of two disk drives. One disk drive will be dedicated to the "read" operations and the other disk drive will be dedicated to the "write" operations.

4.3.2. CPU Parameters

We have assumed that the basic arithmetical operations in the innermost loops of the algorithms are either "Inner Product" or "AXPY" operations:

Inner Product : $a = a + y_i \cdot x_i$

AXPY : $y_i = a \cdot x_i + y_i$

We have taken the unit of execution in the basic step for both of these functions equal to a Tflop, which is the time of a "floating point operation" [DONG79]. A Tflop involves the execution time of a double precision floating point multiplication, a double precision floating point addition, some indexing operations and memory references. Our own experiments and the timing indicated in [DONG83], suggest that, 0.5 to 25 μ seconds (microseconds) per flop is a reasonable range of processing speeds. In the cost equations we shall use the functions T_{AXPY} and T_{INN}, to indicate the

times for an AXPY and inner product operation, respectively. In this paper these functions are assume to be identical. For n floating point operations we have:

$$T_{AXPY}(n) = T_{INN}(n) = n \cdot Tflop$$

4.4. Data Model Parameters

Since the relational and the transposed storage organizations are being compared it is imperative to define the parameters of each. The data set is assumed to be dense with 100 attributes and 230,000 tuples. All the attributes are assumed to be eight byte numbers. With the disk parameter values specified above, there are 23 tuples per page. In the transposed organization, each page will contain 2,383 elements. To simplify the subsequent analysis, we have assumed 2,300 elements are stored in each page of a transposed file. Hence each attribute will occupy 100 pages. Thus in both the relational organization and the transposed organization the data matrix will occupy 10,000 pages. The parameters used are:

R	number of pages occupied by the data set X	10,000 pages
N	number of pages for a column(transposed)	100 pages
w	number of attributes	100
n	number of tuples/observations	230,000
q	number of elements per page	2,300
p	number of active columns for the matrix operations the values considered are 4 to 100 active columns	
M	the size of the main storage the values considered are 10 to 200 pages	

5. X' X

One common method for solving linear least squares problems is to evaluate X' X and proceed with the normal equations. The stability and the tradeoffs of this method compared with orthogonal triangularization methods (such as the QR decomposition and the Singular Value

Factorization) is beyond the scope of this paper (see for example [KENN80]). In Section 5.1 we present an algorithm for a direct implementation of X' X. In Section 5.2 we discuss the performance of the direct algorithm and compare its performance with the vector building block and vector matrix implementations of X' X. Here we have assumed the diagonal elements of X' X (which correspond to the 2-norm squared of the attributes) have already been evaluated.

5.1. The Horizontal Stripes Algorithm

There are three types of algorithms for evaluating X' X. These correspond to the vector building block, vector-matrix, and direct implementations of X' X and will be labeled VBB, VTM, and, ST (for stripes) respectively. In the vector building block (VBB) algorithm, X' X is evaluated as $\frac{p \cdot (p-1)}{2}$ inner products. With the vector-matrix (VTM) algorithm, X' X is evaluated as (p - 1) "vector times matrix" operations. The "vector" of the k^{th} operation is the k^{th} column, and the matrix is the remaining (p - k) columns. With the stripes algorithm (ST), after reading a "horizontal stripe" of the active columns, the partial inner products are accumulated before processing the next stripe. This algorithm involves the fewest page transfers, since the active columns of X are read exactly once. The only disadvantage of the horizontal stripes algorithm is that it requires O(p) pages of main storage.

The details of the vector building block and vector matrix algorithms are given in [KHOS84]. Here we shall only describe one version of the horizontal stripes algorithm. The execution of the algorithm proceeds as follows:

```
ST:
    $ q := 0
      For k := 1 to N / (M / (p + 1))
            For i := 1 to p
                If Find (M / (p + 1)) then Read ( X_i^(M/(p+1)),k )
                set (A [q,i] )
            q := (q+1) mod 2
    $ clear (A) ; q:= 0
      For k := 1 to N / (M / (p + 1))
            clear (D)
            DONE := 0
            For j := 1 to p [ count [j] := 0 ]
            Repeat
                ( i , j ) := FindPair ( A [q ,] , D )
                c_ij := c_ij + X_i^(M/(p+1)),k . X_j^(M/(p+1)),k
                count [i] := count [i]+1; count [j] := count [j]+1
                If count [i] = (p - 1) then Free ( X_i^(M/(p+1)),k )
                if count [j] = (p - 1) then Free ( X_j^(M/(p+1)),k )
                set ( D [i, j] )
                DONE := DONE + 1
            Until DONE = p · (p - 1) / 2
            clear ( A [q ,] )
            q := (q+1) mod 2
```

In steady state, M/(p+1) page blocks of the next horizontal stripe are read while processing the current stripe. The i^{th} position of A is set as soon as the current M/p pages of X_i are read. The function "FindPair" accesses a shared linear array A (of size p) and a two dimensional array D (p by p) and finds two indexes i and j, such that the current M/p page blocks of X_i and X_j are resident and the inner product of X_i and X_j are not yet accumulated. It waits and does not

return until it finds such a pair. For each active column, a counter is incremented when an inner product of the active column is accumulated. When the counter achieves the value p - 1, the block which holds the M/(p+1) pages of the column is released. Since there are p + 1 subdivisions, there can be blocks from at most two horizontal stripes. Finally, the boolean function "Find" attempts to find a free block of M/(p+1) pages, and does not return until it finds one.

5.2. Cost Functions

In the previous section we described a direct algorithm for evaluating X' X with a fully transposed secondary storage organization. In this section we give the cost function for this algorithm and compare its performance with the VBB and VTM algorithms for the fully transposed organization and the stripe-wise algorithm for the relational organization. The cost functions for VBB and VTM algorithms are given in [KHOS84]. To evaluate the total execution times of ST, suppose that there exists an integer k such that:

$$T_{INN}((k+1)\cdot U\cdot q) \geq T_{r\text{-}io}(U) \geq T_{INN}(k\cdot U\cdot q)$$

where $U = \frac{M}{p+1}$. We have two cases:

Case 1: $k \geq \frac{(p-1)}{2}$. It is clear that the algorithm will be I/O bound and the execution time is:

$$p\cdot\frac{N}{(M/(p+1))}\cdot T_{r\text{-}io}\left(\frac{M}{p+1}\right) + \left[(p-1) + \frac{(p-m-1)\cdot(p-m-2)}{2}\right]\cdot T_{INN}\left(q\cdot\frac{M}{p+1}\right)$$

where here $m = \min(k, p-1)$.

Case 2: $k < \frac{(p-1)}{2}$. Here the execution is computation bound and the total execution time is:

$$(k+1)\cdot T_{r\text{-}io}\left(\frac{M}{p+1}\right) + \left[\frac{N}{M/(p+1)}\cdot\frac{p\cdot(p-1)}{2} - \frac{k\cdot(k-1)}{2}\right]\cdot T_{INN}\left(q\cdot\frac{M}{p+1}\right)$$

In Figure 5.1 the total execution time for each algorithm as a function of the number of active columns is presented. The time per floating point operation is assumed to be 0.5 microseconds and the main storage size 100 pages. We have also included the curve for the total execution time of a direct algorithm with the relational storage organization. The VBB algo-

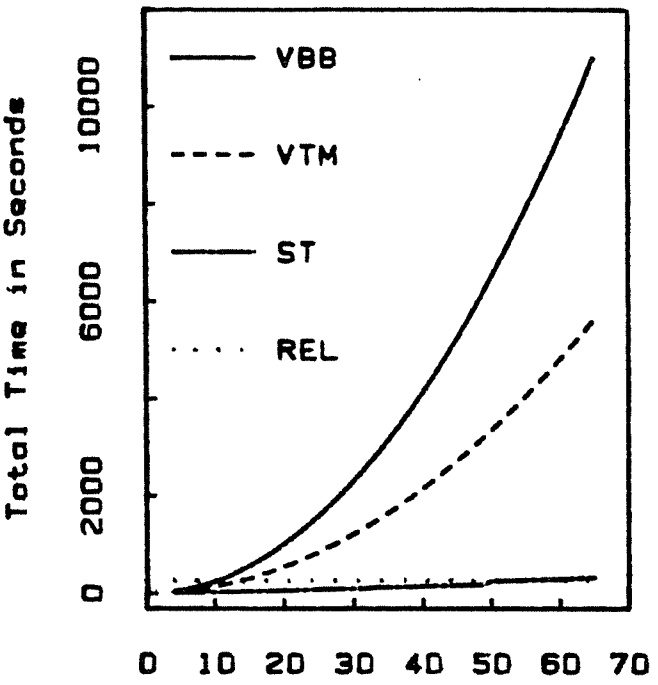

Fig. 5.1

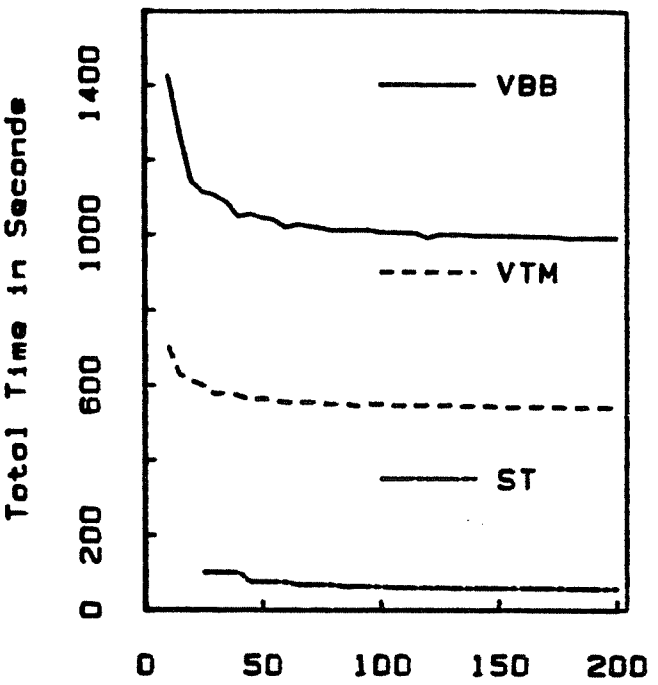

Fig. 5.2

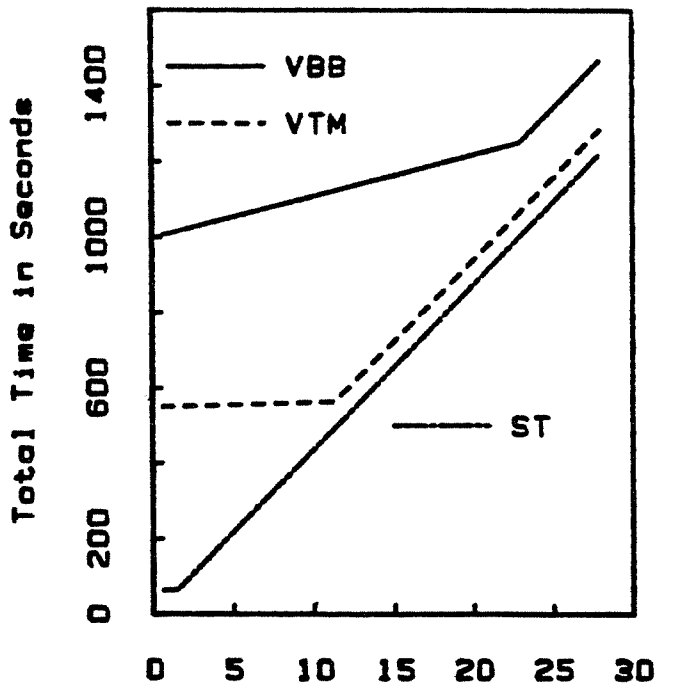

Fig. 5.3

rithm performs better than the relational algorithm only when $p \leq 10$. VTM is better than the relational storage organization when $p \leq 13$ and ST when $p \leq 51$. However, the ST with the transposed file organization always (that is for all values of $p \leq w$) does fewer page transfers than the direct stripe-wise algorithm with the relational storage organization. Since a 0.5 microsecond per flop is an I/O bound case, this shows that seek operations constitute a substantial percentage of the total I/O time for the ST algorithm.

Figure 5.2 presents total execution time curves as a function of main storage size. Here the number of active columns and the time per flop are fixed at $p = 20$ and Tflop $= 0.5$ microseconds. The curves can be approximated by:

$$F(M) = C_1 + \frac{C_2}{M}$$

where C_1 models the transfer time, and the term $\frac{C_2}{M}$ models the time spent during random disk seeks.

Figure 5.3 presents the total execution times for all the algorithms in terms of the time per floating point operation (which is inversely proportional to the CPU speed). The main storage size is 100 pages and there are 20 active columns. The curves clearly possess an I/O bound region and a CPU bound region. The critical value for the stripes algorithms is approximately 1.5 microseconds per flop and for the other algorithms it is about 10 microseconds. The curves clearly illustrate that the stripes algorithm become CPU bound much sooner than the others.

6. The QR Decomposition

For the QR decomposition of X, we apply p Householder transformations on X to reduce it to upper triangular form. Each Householder transformation is defined through a vector "u" and a scalar "c". In fact if H is a Householder transformation then

$$H = I - \frac{u \cdot u'}{c}$$

where I is the n x n identity matrix, u is an n dimensional vector and c is a constant. The u and the c of the k^{th} Householder transformation are functions of X_k (the k^{th} column of X). After the

k^{th} transformation is applied, $X(k,k) = -\text{sign}(X(k,k) \cdot ||x_k||$ and $X(j,k) = 0$ for $j = (k+1), (k+2), \ldots, n$. The other columns are transformed according to:

(a) $t = \dfrac{u' \cdot X_j}{u(1)}$ and

(b) $X_j = X_j - t \cdot u$

Let H1, H2, ..., Hp be the p Householder transformations applied (on the left) to X, and let $Q = H1 \cdot H2 \cdots Hp$. Then

$$Q' \cdot X = R \text{ or } X = Q \cdot R$$

Since $n >> p$, the last (n - p) rows of R are zeros. So for R, the storage requirement is of order $O(p^2)$. Q (or Q') can be recovered through the u's and c's of the Householder transformations. Therefore, rather than storing Q (which is of order $O(n^2)$) explicitly, we can store the u's in the zeroed lower triangular part of X. Whenever Q is needed, the Householder transformations can be applied to the n x n identity matrix. In most applications only the first p columns of Q are needed. If that is the case the Householder transformations are applied to the first p columns of the n x n identity matrix. This is one scheme for accumulating Q or a submatrix of Q. In the next section we shall introduce a more efficient way for accumulating the transformations and constructing Q (or the first p columns of Q). As we shall see, our method will imply a substantial reduction in the I/O overhead.

Throughout the following sections, the column that determines the transformation is called the "pivot," and the remaining active columns are the non-pivotal columns. Our performance evaluation assumes a simple "pivoting scheme," where the pivot at the i^{th} iteration is the i^{th} column. A stabler pivoting scheme is to choose the pivot as the column with the largest norm among the currently active columns. The 2-norms of the columns could be updated (vs. recalculated). Therefore it is possible to determine the column with the largest norm, before the transformation is applied. The expression for the update and the sensitivity analysis is given in [DONG79]. Pivoting on the column with the largest norm simply entails the permutation of some of the columns of X. The performance evaluation of this alternative is essentially the same as for the simple pivoting scheme.

6.1. The Look-Ahead Algorithm

The vector building block (VBB) implementations of the QR decomposition utilize two vector operations: (1) inner product and (2) the AXPY operation. Each of these vector operations is evaluated $O(\frac{p^2}{2})$ times. Similarly for the vector-matrix (VM) strategy there are two main steps: (1) accumulation of the inner products of the pivot with the remaining columns; (2) application of the current Householder transformation. Each is evaluated O(p) times. However, for the k^{th} iteration, the matrix of active columns is n by (p-k). The details of these strategies and their cost equations are given in [KHOS84]. Here we shall primarily concentrate on the direct implementation of the QR decomposition.

To minimize the I/O overhead, we have introduced a look-ahead (LA) scheme which eliminates the I/O references for accumulating the inner products. In the usual implementation of the QR decomposition a pass is made to accumulate the inner products, and a second pass to apply the transformation. With our technique we retain in primary storage the corresponding pages of the current pivot and the next pivot and accumulate the dot products for the next transformation while applying the current transformation. Therefore, except for the first transformation, no secondary storage accesses are made for the accumulation of the inner products.

For the LA algorithm main storage is divided into five subdivisions: (a) M/5 pages of the current pivot; (b) the corresponding M/5 pages of the next pivot; (c) the corresponding M/5 pages of the current column being transformed; (d) the corresponding M/5 pages of the previous column being written back to secondary storage; (e) the corresponding M/5 pages of the previous column being read. Only one iteration of the algorithm is presented.

LA:

Read ($X_i^{M/5,1}$, $X_{i+1}^{M/5,1}$)

For k := 1 to N / (M / 5)

For j := i+1 to p

/* Read Process */
$ If j < p Read ($X_{j+1}^{M/5,k}$)

else If k < N / (M / 5) Read ($X_i^{M/5,k+1}$)

h'14' /* Apply the current Transformation */
$ $X_j^{M/5,k} := X_j^{M/5,k} - t_j \cdot X_i^{M/5,k}$

If j > i+1

/* Accumulate Inner Products For the next Transformation */
$t_{2j} := t_{2j} + X_{i+1}^{M/5,k} \cdot X_j^{M/5,k}$

/* Write the updated block */
$ If j > i+1 Write ($X_{j-1}^{M/5,k}$)

If k < N / (M / 5)

$ Read ($X_{i+2}^{M/5,1}$)

$ Write ($X_p^{M/5,k}$)

6.2. Cost Functions

The total execution time of LA is:

$$T_{LA} = T_{VTM}(p) + \sum_{k=4}^{p} T_{LA}(k, 5) + T_{LA}(3, 4) + T_{ot\text{-}AXPY}$$

where, $T_{VTM}(p)$ is the time to form the inner product of the first pivot with the remaining columns, and $T_{LA}(k, b)$ is the time of a look-ahead pass with k currently active columns. This function is given by:

$$T_{LA}(k, b) = \frac{b \cdot N}{M} \cdot \left[(k-2) \cdot \text{Max} \left(T_{AXPY}(q \cdot \frac{M}{b}) + T_{INN}(q \cdot \frac{M}{b}), T_{r\text{-}io}(\frac{M}{b}) \right) \right]$$

$$+ 2 \cdot T_{r\text{-}io}(\frac{M}{b}) + \frac{b \cdot N}{M} \cdot \left[\text{Max} \left(T_{AXPY}(q \cdot \frac{M}{b}), T_{r\text{-}io}(\frac{M}{b}) \right) + T_{r\text{-}io}(\frac{M}{b}) \right]$$

One basic observation is that with the QR decomposition there is an iterative decrease in the number of active columns. Therefore, in general, the algorithms with the transposed secondary storage organization performed better than the direct algorithm with relational secondary storage organization. In fact the LA algorithm in combination with the transposed secondary organization performed better than the direct implementation with the relational secondary storage organization, for all values of the number of active columns, the main storage size, and the time per flop.

Figure 6.1 contains plots of the total execution time as a function of the number of active columns. The time per flop is 0.5 microseconds and the size of the primary memory is held at 100 pages. The VBB algorithm outperforms the algorithm for the relational organization, when $p \leq 10$, and the VM algorithm outperforms the algorithm for the relational organization for $p \leq 41$.

Figures 6.2 and 6.3 present the total execution time for each algorithm as a function of main storage size. The time per flop is 0.5 microseconds in Figure 6.2 and 25 microseconds in Figure 6.3. The number of active columns is 20. The curves for VBB, VM, and LA possess minimums in Figure 6.3. It is possible to obtain approximate parametric equations of M as a function of the system parameters, by solving minimization problems for each of VBB, VM and LA. The minimization problem will determine the minimum value of the utilized main storage size M. The details and the algorithms for evaluating the optimal values of M are given in [KHOS84].

The total execution time as a function of the time per flop for each algorithm is plotted in Figure 6.4. The critical points in these curves are those points where the "Max" terms in the expressions for the total execution times become CPU bound. The critical point of LA corresponds to the value of Tflop which makes the term

$$\text{Max}\ (\ T_{AXPY}\ (q \cdot \frac{M}{5}) + T_{INN}(q \cdot \frac{M}{5}),\ T_{r\text{-}io}(\frac{M}{5}))$$

CPU bound, that is Tflop $\approx$ 6 microseconds.

As in the case of X'X, the LA algorithm becomes CPU bound with smaller values of Tflop.

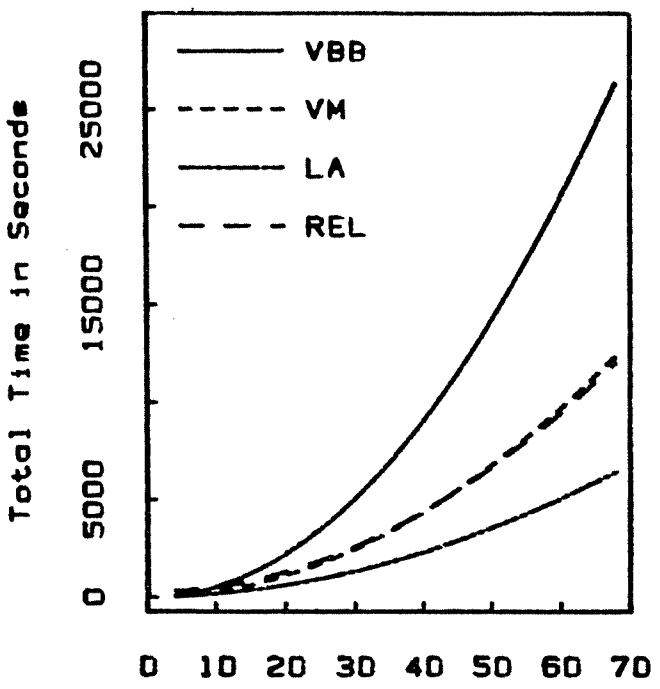

Fig. 6.1

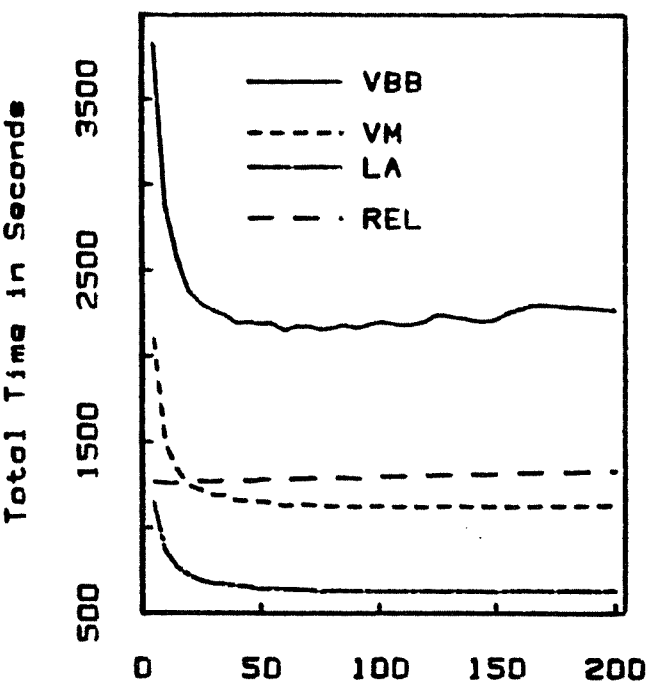

Fig. 6.2

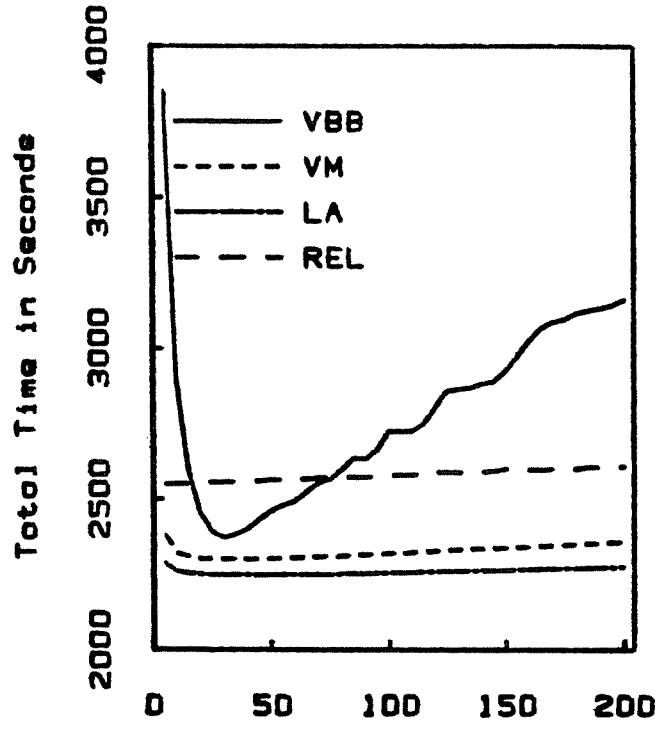

Fig. 6.3

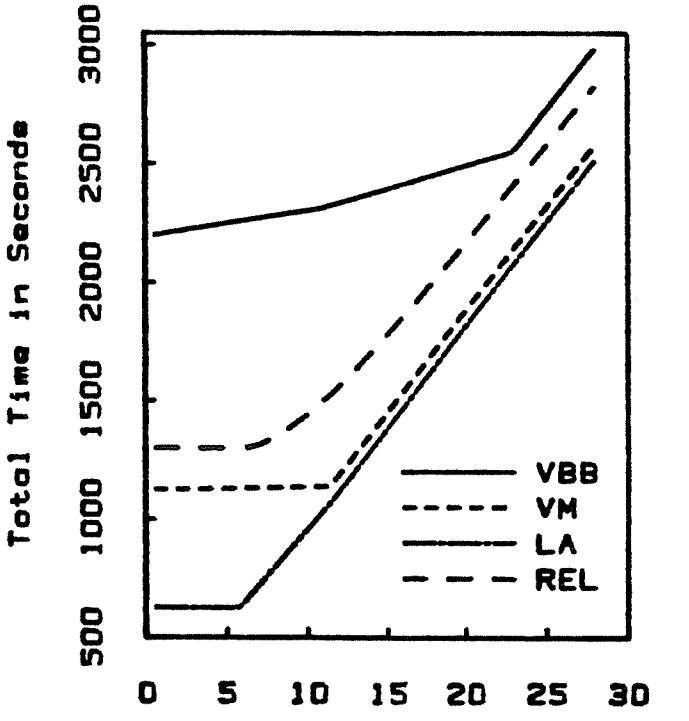

Fig. 6.4

Rendering the execution of an operation on a large data set CPU bound is one way of solving the I/O problem.

7. The Singular Value Factorization

The Singular Value Factorization attempts to compute an n by p matrix W, a p x p orthogonal matrix V, and a p x p diagonal matrix D such that

$$X = W \cdot D \cdot V'$$

The p columns of W are the eigenvectors associated with the p largest eigenvalues of XX' . The columns of V are the orthonormalized eigenvectors of $X'X$. The non-ascending ($s_1 \geq s_2 \geq s_3 \geq \cdots \geq s_p$) diagonal elements of D are the non-negative square roots of the eigenvalues of $X'X$ (also called the "singular values" of X). Both D and V are of order O (p^2), and therefore are stored in primary memory. W, however, is of order O($n \cdot p$), and should be stored on disk. With the Golub-Reinsch algorithm [WILK71], the decomposition proceeds in two main steps:

Step 1 - **Reduction to Bidiagonal Form:** This is accomplished by premultiplying X with p Householder transformations and postmultiplying it with p - 2 Householder transformations.

Step 2 - **Decomposition of the Bidiagonal Matrix:** This is done through the implicitly shifted QR algorithm [WILL71, STEW73].

For large matrices, several modifications to the Golub-Reinsch algorithm have been proposed. Chan [CHAN82] suggests first to triangularize the matrix X using Householder transformations and then apply an SVF algorithm to the n by n upper triangular matrix. Cuppen [CUPP81] points out that if the "ultimate shift" strategy is used instead of the implicit shift, the actual number of sweeps per eigenvalue is reduced to one. Therefore, theoretically, we can guarantee that the amount of storage for the rotations will be O (p^2).

In the following we shall assume Chan's approach, as well as a p by p matrix L which contains the accumulated rotations. We shall also assume the u's of the left Householder transformations are stored in the zeroed subdiagonal part of X.

Next, let us illustrate how it is possible to accumulate the left Householder transformations and construct W, if the inner product of the u's and the first p elements of each u are available. Suppose we want to evaluate:

$$Q = (I - \rho_1 u_1 \cdot u_1').(I - \rho_2 u_2 \cdot u_2') \cdots (I - \rho_p u_p \cdot u_p')$$

It can easily be shown that:

$$Q = I + \sum_{i=1}^{p} \sum_{j=i}^{p} c_{i,j} u_i \cdot u_j'$$

where the $c_{i,j}$'s are in terms of the ρ_i's and the inner products of the u's. The first p elements of the k^{th} row of $u_i \cdot u_j'$ is given by:

$$u_{i,k} \cdot [u_{j,1}, u_{j,2}, \cdots , u_{j,p}]$$

where the first (j-1) elements of u_j are zero. Let U be the matrix of the u's and Q_p the first p columns of Q. Let T be given by:

$$T_{i,j} = \sum_{k=i}^{j} c_{i,k} \cdot u_{k,j} \ ; \ 1 \leq i \leq j \leq p$$

Then the rows k through m of Q_p can be obtained by accessing the rows k through m of U, and performing a matrix multiplication of the (m-k)+1 by p submatrix of U with the upper triangular matrix T. If $k \leq p$ we need to add 1 to the diagonal elements to this matrix product. Therefore,

$$W = Q_p \cdot L = (I_p + U \cdot T) \cdot L = I_p \cdot L + U \cdot (T \cdot L)$$

and W could be obtained basically through the product of the n by p matrix U with the p by p matrix $T \cdot L$.

7.1. Direct Implementation

We shall describe only the step for constructing W. The vector building strategy utilizes the inner product, AXPY, and "scale" vector operations.[2] The vector-matrix approach utilizes vector-times-matrix, Householder transformation, and matrix-times-times vector steps for the construction of W. Here we shall present a direct algorithm for constructing W.

The main storage subsystem is divided into $2 \cdot (p + 1)$ equal subdivisions. Let $B = \frac{M}{2 \cdot (p + 1)}$. Initially p B-page blocks are allocated to the first stripe of W. There are

three concurrent processes.

(1) The "Read"" process finds a free block (of size B pages) and reads the next B pages of the next column of U. Once the B pages of a column are read, the process sets the corresponding entry in the two dimensional boolean array A.

(2) The "Accumulate" process first checks if A [i,k] is set. If so, it accumulates the contributions of the corresponding pages of U_i to the current stripe of W. If k = 1 and $U_i^{B,1}$ is processed, the B pages of U_i are allocated to the accumulation of the next stripe of W through the function "Alloc." Otherwise (that is $k \neq 1$), the B pages of U_i are freed. When stripe k of W is constructed, the corresponding entry in the boolean area "Avail" is set.

(3) The "Write" process first allocates the pages of stripe k of W to the accumulation of stripe k+1 of W. Next it checks if the current (or k^{th}) block of W is available to be written, and, if so, writes it to secondary storage.

[2] This last operation multiplies a vector with a scalar value.

DIR:

$$B = \frac{M}{2 \cdot (p + 1)}$$

Alloc ($M^{p \cdot B}$, $W^{p \cdot B, 1}$)

For k := 1 to N/B

$ For i := 1 to p /* the Read process */

If Find (B) then Read ($U_i^{B,k}$)

set (A [i,k])

$ For k := 1 to N/B /* the Accumulate Process */

For i := 1 to p

Check (A [i,k])

For j := 1 to p

$W_j^{B,k} := W_j^{B,k} + U_i^{B,k} \cdot (TL)_{i,j}$

If (k = 1) then Alloc ($U_i^{B,k}$, $W^{p \cdot B, 2}$)

else Free ($U_i^{B,k}$)

Set (Avail [k])

$ For k := 1 to N/B /* Write process */

If k < (N/B – 1) then

Alloc ($W^{p \cdot B,k}$, $W^{p \cdot B,k+2}$)

Check (Avail [k])

Write ($W^{p \cdot B, k}$)

7.2. Cost Functions

The proposed approach presents a new method for evaluating the SVF of a data matrix X, which consists of three main steps: (1) the QR decomposition of X, (2) the evaluation of U' U , and (3) the matrix product step for forming W. For the cost equation of the direct algorithm we need to distinguish between the I/O and CPU bound cases. Let $B = \frac{M}{2 \cdot (p + 1)}$. If

T_{AXPY} (B · q · p) > $T_{r\text{-}io}$(B)

then the algorithm will be CPU bound, otherwise it will be I/O bound. The total execution time for the I/O bound case is:

$$p \cdot (2 \cdot (p + 1)) \cdot \frac{N}{M} \cdot T_{r\text{-}io}(B) + p \cdot T_{r\text{-}io}(B) + T_{AXPY}\,(\,B \cdot q \cdot p\,)$$

and for the CPU bound case is:

$$(p + 1) \cdot T_{r\text{-}io}(B) + T_{AXPY}\,(\,N \cdot p^2 \cdot q)$$

An important observation here is that with a direct implementation the construction of W requires only O (1) passes over an n by p matrix. This constitutes a substantial savings in data accesses compared to the more conventional method of constructing W. The usual approach is to apply the Householder transformations to the first p columns of the n by n identity matrix (forming Q_p) and then apply the left rotations to Q_p. This would require at least O (p) passes over an n by p matrix. With the relational secondary storage organization, the accumulation of the inner products of the u's (the U' U step), was incorporated in the QR decomposition step. Consequently, the construction of W required just one pass over the n by p matrix U.

With the transposed secondary storage organization, it is important to analyze the contribution of the QR decomposition and U' U steps to the cost functions. For the percentage of the total I/O as a function of the number of active columns, the QR decomposition step involved about 38% percent of the total I/O and the U' U step about 15%, with the VBB strategy. The corresponding percentages for the VM strategy were, respectively, 45% and 15%. However, for the direct implementations, the percentage of the QR decomposition step went up from about 65% to 93% when the number of active columns was varied from 5 to about 70. The percentage of the U' U step went down from about 11% to 2%. Table 7.1 summarizes these percentages.

Strategy	QR	U' U
VBB	38%	15%
VM	45%	15%
DIR	65% to 94%	11% to 2%

Table 7.1

Therefore, the QR step constitutes a substantial percentage of the total I/O for the algorithms with the transposed organization. For the direct implementations it is at least 65% of the total I/O.

Another important observation is that with the Singular Value Factorization we observed the cumulative effect of the implementations at the three abstraction levels. This was due to the implementation of the QR decomposition step, the U' U step, and the matrix product step all at the same abstraction level. Table 7.2 summarizes the threshold values of p for X'X, QR and SVF.

Algorithm	X'X	QR	SVF
Vector Building Block	10	10	5
Vector-Matrix	13	41	7
Direct Implementation	51	all p	all p

Table 7.2

The threshold values are the maximum number of active columns for which the algorithms with the transposed organization outperform the direct implementations with the relational organization. These values are for the I/O bound total execution times. Notice the optimality of the direct implementations with the transposed organization, especially for the Singular Value Factorization and the QR decomposition. The cumulative effect of implementations at levels one (VBB) and two (VM) is significant.

8. Conclusions

Data management issues relating to statistical databases have received growing attention in recent years. Although several interesting solutions to the characteristic problems of large statistical databases have emerged, the data management issues of the computational methods have been generally ignored. To support statistical operations efficiently, this paper endorsed an integrated approach to statistical databases. With an integrated system, the database management supports both the data intensive and computational queries.

The main emphasis has been the comparative performance evaluation of three important computational methods: $X'X$, the QR decomposition, and the Singular Value Factorization. The alternative algorithms were compared with respect to two basic types of secondary storage organizations (relational and transposed) as well as three abstraction levels corresponding to vector building block, vector-matrix, and direct implementations. The basic contributions of our research can be summarized as follows:

(1) we developed closed form equations, for the I/O costs and the total execution times in terms of the system and data parameters. The variable parameters were the number of active columns, main storage size, and time per floating point operation.

(2) we analyzed the effects of the transposed and relational secondary storage organizations on the total I/O and the total execution time.

(3) we performed a comparative evaluation of alternative algorithms for vector building block, vector-matrix and direct implementations. The algorithms explicitly specify the buffer management strategy that provides optimal performance.

The performance evaluation assumed a simple but general system architecture. The generality of the system architecture enables the designers of the database management system to modify the parameter values to suit their particular architectures. This, in turn, would facilitate choosing between the transposed and relational secondary storage organizations and also the abstraction level of the algorithms to be supported.

The vector building-block and vector-matrix implementations of the computational methods performed considerably poorer than the direct implementations. For example, with $X'X$ and the matrix product step of SVF, the vector building block and vector matrix strategies involved $O(p^2 \cdot N)$ page transfers. However, with the direct implementations, only $p \cdot N$ page transfers are needed for these operations. Therefore, an integrated system supporting the computational

methods through vector buiding blocks and vector-matrix operations will perform considerably poorer than an integrated system which supports the computational methods through direct implementations.

The options to a building blocks approach for an integrated statistical database system are either integrated systems where the computational methods are supported through direct implementations, or interface systems. The former option implies an implementation per computational method. The problem with this approach is that the set of statistical computational methods is continually expanding. Whenever, a new computational method is introduced, the database management system software must be extended to include a direct implementation of the new computational method.

The interface system approach is not likely to yield a satisfactory solution as most statistical and linear algebra packages implement the computational methods through vector building blocks. Furthermore, these packages utilize the global buffer management strategy of the underlying operating system.

Another important conclusion deduced from this study concerns the relative performance of the transposed and relational storage organizations. The direct implementation of each algorithm using the transposed secondary storage organization were faster than the direct implementation that used the relational storage organization when the number of active columns was less than or equal to 50 in the case of $X^t X$ and for all ranges of active columns in the case of the QR decomposition and the Singular Value Factorization. In fact, in the case of the QR decomposition and the Singular Value Factorization, increasing the number of active columns also increased the difference between the cost functions[3] of the transposed and relational direct algorithms, in favor of the transposed organization. Therefore, our results suggest that for those statistical computational methods whose algorithms involve an iterative decrease in the number of currently active columns, the underlying storage structure must be the fully transposed secondary storage organization.

---

[3] both I/O and total execution times

Although the transposed and the relational organizations are the main secondary storage layouts commonly utilized in most existing statistical database management systems,[4] it might be worthwhile to investigate other alternative storage structures. As a simple example, assume that the users of the system (i.e. the analysts) are capable of predetermining the attribute subsets which they will analyze. If it is feasible to convey this suggestive information to the database management system, then it becomes possible to construct a clustering of the attribute sets and store the data sets as partially transposed files. This will reduce the total I/O time for $X'X$ type operations, as well as the overhead of the initial projection step for those operations that produce temporary files (e.g. the QR decomposition).

Other avenues of future research appear fruitful. The first, is the development of a comprehensive list of computational methods. The computational methods analyzed were those of multiple linear regression. Although the QR decomposition and the Singular Value Factorization are used in other important statistical techniques (e.g. canonical correlation and principal component analysis), there are a number of other statistical methods, with corresponding computational methods, whose data management problems need to be explored and analyzed. The goal here is to provide the analyst with a comprehensive list of alternative statistical techniques, which have high performance and which also have the capability of manipulating large data sets. To this end we need to pursue the development of a list of the most frequently used statistical tools and, correspondingly, the most useful computational methods.

To date, research has mostly been "algorithm" directed. The underlying system architecture for each of the performance evaluations was simple and the emphasis was on alternative algorithms for this general type of architecture. In [KHOS84] we proposed special purpose multiprocessor architectures for the direct algorithms of the computational methods. Another avenue of research would be to propose secondary storage organizations and parallel algorithms for existing loosely coupled multiprocessor architectures. In other words, it could be worthwhile to investigate the performance issues of distributed statistical databases. A novel area here is the analysis

[4] for example SEEDIS [McCA83] uses variable length records, whereas RAPID [TURN79] uses fully transposed files

of distributed algorithms for a number of computational operations (e.g. X' X, QR, SVF etc.), over large and distributed data sets.

BIBLIOGRAPHY

[BAKE76] Baker, M., "Users's Guide to the Berkley Transposed File Statistical System: PICKLE," Technical Report No. 1, 2nd ed., University of California, Berkley Survey Research Center, 1976.

[BATO79] Batory, D. S., "On Searching Transposed Files," ACM Transactions on Database Systems, Vol. 4, No. 4, December, 1979.

[BELL83] Bell, J., "Data Structures For Scientific Simulation Programs'" Proceedings of the Second LBL Workshop on Statistical Database Management, September, 1983.

[BORA82] Boral, H., DeWitt, D. J., and Bates, D. M., "Framework for Research in Database Management for Statistical Analysis," University of Wisconsin-Madison, Computer Sciences Tech. Report #465, February, 1982.

[BURN81] Burnett, R., and Thomas, J., "Data Management Support for Statistical Data Editing," Proceedings of the First LBL Workshop on Statistical Database Management, December, 1981.

[CHAN82] Chan, T. F., "An Improved Algorithm for Computing the Singular Value Decomposition," ACM Transactions on Mathematical Software 8, 1, 1982.

[CUPP81] Cuppen, J. J. M., "The Singular Value Decomposition in Product Form," Mathematisch Institut, Amsterdam, pp. 81-106, April, 1981.

[DENN83] Denning, D., Nicholson, W., Sande, G., and Shoshani, A., "Research Topics in Statistical Database Management," Proceedings of the Second International Workshop on Statistical Database Management, September, 1983.

[DONG79] Dongarra, J., Moler, C., Bunch, J., and Stewart, G., Linpack Users' Guide, SIAM, 1979.

[DONG83] Dongerra, J., "Redesigning Linear Algebra Algorithms," Bulletin De La Direction Des Etudes Et Des Rescherches, Serie C, No 1, 1983.

[EGGE81] Eggers, S. J., Olken, F., and Shoshani, A., "A Compression Technique for Large Statistical Databases," Proceedings of the 7th International Conference on Very Large Data Bases, France, 1981.

[ELSH74] Elshoff, James L., "Some Programming Techniques for Processing Multi-dimensional Matrices in a Paging Environment," AFIPS Conference Proceeding, Vol. 43, 1974.

[GOLO67] Golomb, S. W., Shift-Register Sequences, Holden-Day, Inc., San Fransisco, 1967.

[GOLU73] Golub, G., and Styan, G., "Numerical Computations for Univariate Linear Models,"

Journal Statistical Computing, Vol. 2, 1973.

[HOFF76] Hoffer, A. J., "An Integer Programming Formulation of Computer Data Base Design Problems," Information Sciences 11, pp. 29-48, 1976.

[IBM 77] IBM, "Reference Manual for IBM 3350 Direct Access Storage," GA26-1638-2, File No. S370-07, IBM General Products Division, San Jose, California, April, 1977.

[KENN80] Kennedy, W., and Gentle, J., Statistical Computing, Marcel Dekker, Inc., 1980.

[KHOS84] Khoshafian, Setrag, A Building Blocks Approach To Statistical Databases, Ph.D. thesis, University of Wisconsin-Madison, Tech. Rep. #543, May 1984.

[LAWS79] Lawson, C, Hanson, R., Kincaid, D., and Krough, F. "Basic Linear Algebra subprograms for Fortran usage", ACM Trans. Math. Software, 5(3), 1979.

[MAND82] Mandel, J., "Use of Singular Value Decomposition In Regression Analysis," The American Statistician, Vol. 36, No. 1, February, 1982.

[McCA82] McCarthy, J. L., "The SEEDIS Project: A summary Overview," Lawrence Berkley Lab., LBL-14083, 1982.

[McCU83] McCullagh, P., Nelder, J. A., Generalized Linear Models, New York, Chapman and Hall, 1983.

[McKE69] McKellar, A. C., and Coffman, E. G. Jr., "Organizing Matrices and Matrix Operations for Paged Memory Systems," Communications of the ACM, VOl. 12, No. 3, 1969.

[MEYE69] Meyers, E. D. Jr., "Project IMPRESS: Time Sharing in the Social Sciences," AFIPS Conference Proceedings of the Spring Joint Computer Conference, 1969.

[MOLE72] Moler, Cleve B., "Matrix Computations with Fortrans and Paging," Communications of the ACM, Vol. 15, No. 4, 1972.

[PERR81] Perry, J. R., "Secondary Storage Methods for Solving Symmetric, Positive Definite, Banded Linear Systems," Yale University Computer Science Research Report #201, 1981.

[RICE83] John, R. Rice, "PARVEC workshop on Very Large Least Squares Problems and Supercomputers", CSD-TR#464, Purdue, December, 1983.

[ROWE83] Rowe, N. C., "Top-Down Statistical Estimation on a Database," Proceedings of SIGMOD 83, 1983.

[RYAN76] Ryan, T. A., Joiner, B. L., and Ryan, B. F., MINITAB Student Handbook, Duxbury press, Massachusetts, 1976.

[SEBE77] Seber, G., Linear Regression Analysis, John Wiley & Sons, 1977.

[SHOS82] Shoshani, A., "Statistical Database: Characteristics, Problems, and Some Solutions," Proceeding of the Eight International Conference on VLDB, 1982.

[STAN83] Stanley, S. Y. W., Navathe, S. B., and Batory, D. S., "Logical and Physical Modeling of Statistical/Scientific Databases," Proceedings of the Second International Workshop on Statistical Database Management, 1983.

[STEW73] Stewart, G., Introduction to Matrix Computations, Academic Press, 1973.

[THOM83] Thomas, J. J., and David, H. L., "ALDS Project: Motivation, Statistical Database Management Issues, Perspectives, and Directions," Proceedings of the Second International Workshop on Statistical Database Management, 1983, pp. 82-88.

[TRIVE77] Trivedi, Kishor S., "On the Paging Performance of Array Algorithms," IEEE Transaction on Computers, Vol. c26, No. 10, 1977.

[TUKE77] Tukey, J. W., Exploratory Data Analysis, Addison-Wesley, Massachusetts, 1977.

[TURN79] Turner, M. J., Hammond, R., and Cotton, P., "A DMBS for Large Statistical Databases," Proceedings of the 5th International Conference on VLDB, pp. 319-327, 1979.

[WILL71] Wilkinson, J. H., and Reinsch, C., Linear Algebra, Springer-Verlag, 1971.